HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION COMMUNICATION TECHNOLOGY

CAPSTONE PROJECT REPORT:

Story Point Estimation

Supervised by:

Mentor. Dang Yen Trang

Presented by:

Au Trung Phong - 20225455

Hanoi - Vietnam

2024

Contents

1 Introduction **2**

1.1 Problem's Background 2

1.2 About the dataset 2

2 Preprocessing **4**

2.1 Label preprocessing 4

2.2 Feature preprocessing 5

3 Methodology: Machine Learning-based Approaches **6**

3.1 Linear regression 6

3.2 Support Vector Machine 7

3.3 Random Forest 8

3.4 Gradient Boosting 10

3.5 Stacking Regressor 12

4 Experimental results **14**

4.1 Model selection results 14

4.2 General observations 17

4.3 Compare between Bag of Words and doc2vec 18

5 Conclusion and Further Enhancements **19**

6 Acknowledgement **20**

7 References **21**

1 Introduction

1.1 Problem's Background

In modern agile development settings, software is developed through repeated cycles (iterative) and in smaller parts at a time (incremental), allowing for adaptation to changing requirements at any point during a project's life. A project has a number of iterations (e.g. sprints in Scrum). Each iteration requires the completion of a number of user stories, which are a common way for agile teams to express user requirements.

There is thus a need to focus on estimating the effort of completing a single user story at a time rather than the entire project. In fact, it has now become a common practice for agile teams to go through each user story and estimate its "size". Story points are commonly used as a unit of measure for specifying the overall size of a user story.

In this project, I aim to develop a model that predicts story points based on the task's **Title** and **Description**.

1.2 About the dataset

The dataset is prepared from 16 different projects:

appceleratorstudio	aptanastudio	bamboo	clover
datamanagement	duracloud	jirasoftware	mesos
moodle	mule	mulestudio	springxd
talenddataquality	talendesb	titanium	usergrid

In this section, I will focus on analyzing the general information across all the datasets, rather than examining each one individually, due to the similarity in patterns and trends observed among them. This approach allows for a more streamlined and cohesive analysis without repetitive details from each dataset.

Size: The dataset sizes vary considerably, ranging from 286 to 4,030 instances. The datasets with the largest number of instances are **datamanagement**, **springxd**, and **appceleratorstudio**. On the other hand, some datasets are smaller, with around 300 instances each, including **jirasoftware**, **usergrid**, **clover**, and **bamboo**.

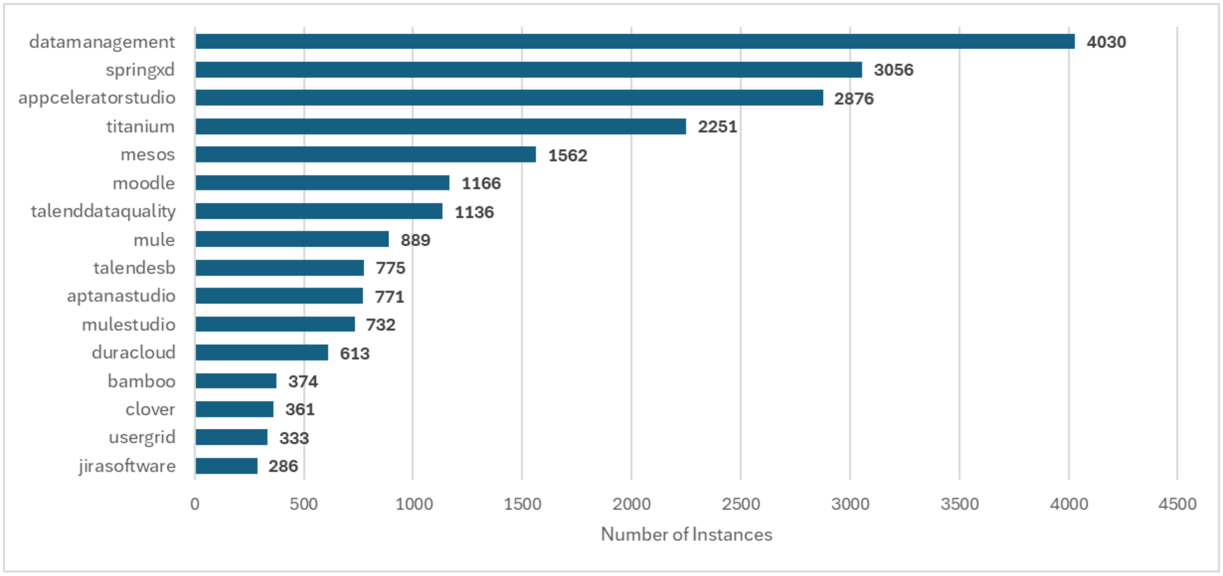

Figure 1: Bar chart of datasets's size

Label analysis:

The value of story points ranges from 1 to 100. However, across the datasets, the average number of unique values is only **12**, with the highest being **20** (in the datamanagement dataset) and the lowest being **7** (in the talendesb dataset). More specifically, the average number of values that account for more than **5%** of instances is **5**, representing about **51%** of the total number of unique values.

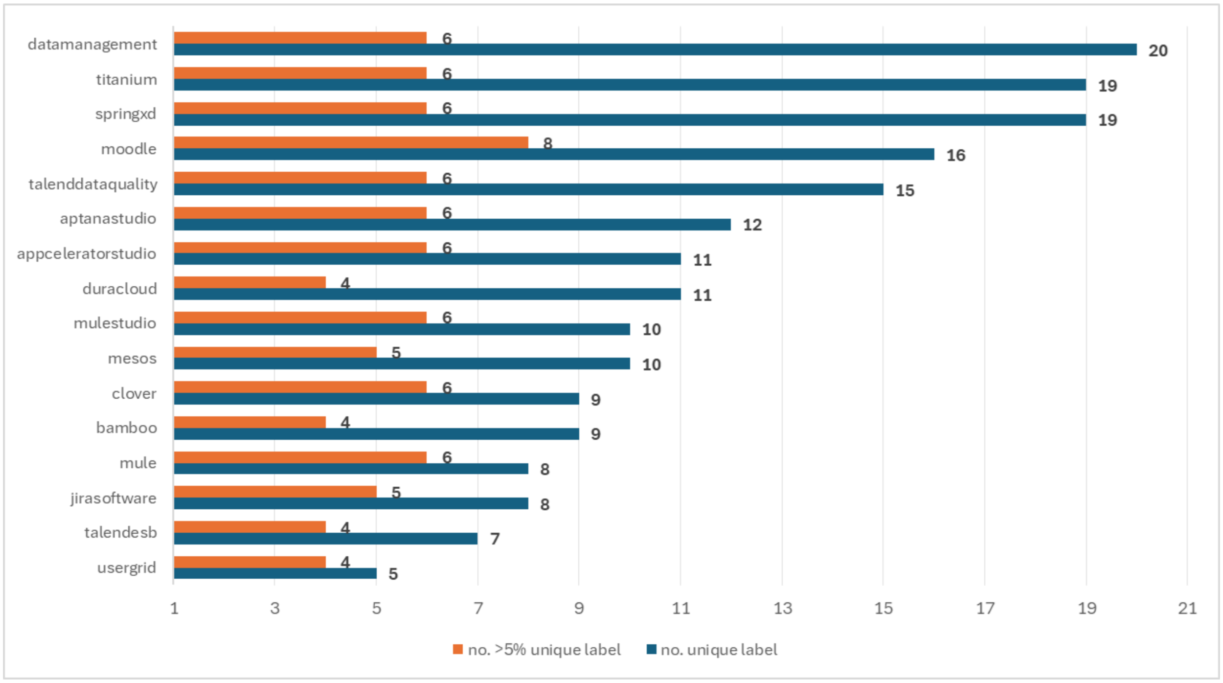

Figure 2: Label count analysis

One more signature of these datasets is about **skewness** and **kurtosis**:

Dataset	Skewness	Kurtosis
appceleratorstudio	2.83	19.85
aptanastudio	2.23	8.87
bamboo	4.05	25.93
clover	3.66	15.93
datamanagement	3.52	13.63
duracloud	3.66	18.41
jirasoftware	2.18	5.95
mesos	4.01	40.69

Dataset	Skewness	Kurtosis
moodle	2.81	8.02
mule	1.27	2.86
mulestudio	2.61	9.44
springxd	3.16	24.24
talenddataquality	2.19	8.53
talendesb	2.48	9.97
titanium	2.18	6.41
usergrid	1.24	2.72

All the datasets have positive **skewness** and **kurtosis** values that are significantly greater than 0. This indicates that the label distributions of all the datasets are highly **right-skewed** and **leptokurtic** (heavy-tailed), suggesting the presence of potential outliers on the right side of the distribution. Therefore, a solution is needed to address these issues, which will be described in the next section.

Title/Description analysis:

My first approach involves analyzing the lengths of titles and descriptions. The distributions of these lengths do not exhibit any significant attributes. However, there is a slight linear relationship between the lengths of titles and descriptions, which could provide a clue for feature engineering in the next section.

I also delved deeper in the titles and descriptions by analysing the most frequent words. However, these words do not exhibit any notable properties; most of them appear to be random words without significant meaning.

2 Preprocessing

2.1 Label preprocessing

To address the issues of **skewness** and **kurtosis**, I have two potential solutions: cut-off and logarithmic transformation.

Cut-off: This approach involves accepting a small amount of data loss (not exceeding 5%) to remove the right-tailed data. While this method can reduce both skewness and kurtosis, it is less prefered due to the potential loss of valuable data.

Logarithm: This solution involves applying a logarithmic transformation to the target value.

This approach can decrease kurtosis and bring skewness closer to zero while maintaining data integrity.

Dataset	Skewness		Kurtosis	
	Before	After	Before	After
appceleratorstudio	2.83	-0.64	19.85	1.12
aptanastudio	2.23	-0.86	8.87	0.62
bamboo	4.05	0.72	25.93	0.35
clover	3.66	0.79	15.93	0.03
datamanagement	3.52	0.64	13.63	-0.16
duracloud	3.66	1.06	18.41	0.57
jirasoftware	2.18	0.11	5.95	-0.19
mesos	4.01	0.23	40.69	-0.35
moodle	2.81	0.20	8.02	-0.46
mule	1.27	-0.47	2.86	-0.73
mulestudio	2.61	-0.18	9.44	0.04
springxd	3.16	0.03	24.24	-0.71
talenddataquality	2.19	-0.23	8.53	-0.69
talendesb	2.48	0.53	9.97	-0.46
titanium	2.18	-0.10	6.41	0.01
usergrid	1.24	-0.58	2.72	0.36

In conclusion, the **logarithmic transformation** is prefered method for addressing skewness and kurtosis and will be used for training models in the upcoming section.

2.2 Feature preprocessing

2.2.1 Bag of Words

The bag-of-words model is a model of text that uses a representation of text that is based on an unordered collection (or "bag") of words. It disregards word order (and thus any non-trivial notion of grammar) but captures multiplicity. This makes BoW models easier for implementation and inference. However, the unordered nature of BoW can frequently lead to the loss of information in the corpus.

In this project, I vectorized both the title and the description of the task into two separate vectors. I utilized unigrams and bigrams to capture more information, including only those words that occur at least twice in the vocabulary. Finally, I concatenated these two vectors to create a comprehensive feature set.

Additionally, based to the linear relationship analysed in the previous section, I decided to include two more features: the length of the title and the length of the description into the final set of features.

2.2.2 doc2vec

Word2Vec is a word embedding technique used to map words to real vectors, capturing the similarity between different words effectively. The Word2Vec tool includes two models: the Skip-gram model and the Continuous Bag of Words (CBOW) model, both of which are self-supervised.

- The Skip-gram model assumes that a word can be used to predict its surrounding words in a text sequence.
- The Continuous Bag of Words (CBOW) model assumes that a center word is predicted based on its surrounding context words.

Doc2Vec extends this approach by mapping entire documents to real vectors, rather than just individual words. It does this by taking the average of the word vectors, which are generated using the Word2Vec technique.

3 Methodology: Machine Learning-based Approaches

I conducted a range of Machine Learning-based algorithms in this section. In each algorithm, we set up a range of corresponding models, each with different hyper-parameters. We then perform a comparison between the models applying the same hyper-parameters to find the best-performing ones using GridSearchCV with cross_validation enabled from Scikit-learn, before comparing those "best" implementations to deliver a broader insight on how would each algorithm interact with the datasets.

3.1 Linear regression

Linear regression is one of the simplest regression algorithms. Assuming a linear relationship between the features and the target values, linear regression aims to fit a hyperplane that maps the features to the target values.

The simplest form of the problem is to find the weights **w** and bias **b** that minimize the following loss function:

$$J(\mathbf{w}, b) = \frac{1}{2N} \sum_{i=1}^{N} (y_i - ({x_i}^T \mathbf{w} + b))^2$$

This optimization can be accomplished using the gradient descent method.

However, linear regression is often combined with regularization of the weights to control the trade-off between model complexity and generalization. In this project, I use three types of regularization: **Ridge**, **LASSO** and **Elastic Net**;

- **Ridge:** adds a squared penalty term to the loss function

$$J(\mathbf{w}, b) = \frac{1}{2N}\sum_{i=1}^{N}(y_i - ({x_i}^T\mathbf{w} + b))^2 + \frac{\alpha}{2}||\mathbf{w}||_2^2$$

- **LASSO:** adds an absolute penalty term to the loss function

$$J(\mathbf{w}, b) = \frac{1}{2N}\sum_{i=1}^{N}(y_i - ({x_i}^T\mathbf{w} + b))^2 + \alpha||\mathbf{w}||_1$$

- **Elastic Net:** combines both the squared and absolute penalty terms, controlled by the l1_ratio parameter.

$$J(\mathbf{w}, b) = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - (\mathbf{x}_i^T\mathbf{w} + b)\right)^2 + \alpha \cdot \text{l1\_ratio} \cdot \|\mathbf{w}\|_1 + \frac{\alpha}{2} \cdot (1 - \text{l1\_ratio}) \cdot \|\mathbf{w}\|_2^2$$

Implementation: We use the following hyperparameters to tune the linear regression model:

- α: constant that controls the weight of penalty term.
- l1_ratio: constant that controls the l1 regularization ratio in Elastic Net regularization.

3.2 Support Vector Machine

The goal of ε - Support vector regression is finding a function $f(x) = x^T\mathbf{w} + b$ that has at most ε deviation from the actually obtained targets y_i for all the training data and at the same time as flat as possible. Therefore, the problem becomes a convex optimization problem:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}||w||_2^2 + C \cdot \sum_{i=1}^{l}(\xi_i + \xi_i^*) \\ \text{subject to} & \begin{cases} y_i - ({x_i}^T\mathbf{w} + b)) \le \varepsilon + \xi_i \\ ({x_i}^T\mathbf{w} + b)) - y_i \le \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0 \end{cases} \end{array}$$

with:

$$|\xi|_\varepsilon = \begin{cases} 0 & \text{if } |\xi| < \varepsilon \\ |\xi| - \varepsilon & \text{otherwise} \end{cases}$$

The constant $C > 0$ determines the trade-off between the flatness of f and amount up to which deviations larger than ε are tolerated:

- When C is low, the scarification makes less impact on the objective function leading the algorithm to maximize the margin (minimize $\|w\|_2^2$) so that $\sum_{i=1}^{l}(\xi_i + \xi_i^*)$ becomes larger.
- When C is high, the algorithm will focus on minimizing $\sum_{i=1}^{l}(\xi_i + \xi_i^*)$ leads the margin becomes harder.

However, there are situations where the problem cannot be addressed effectively using a simple linear model. In these cases, the kernel method can provide a solution. The kernel method generally involves using a function $\phi(x)$ to project the data from the original space to a higher-dimensional space where the data may become more amenable to regression. In this transformed space, Support Vector Regression (SVR) can be applied to find a function that best fits the data.

Due to the computational cost of the kernel function, the kernel trick technique that chooses only the function calculating the scalar product of two data points becomes the solution. Some typical kernel functions are referred to in Table 1.

Name	Formula
Linear	$x^T z$
Polynomial	$(r + \gamma x^T z)^d$
Sigmoid/Tanh	$\tanh(r + \gamma x^T z)$
Radial Basis Function	$\exp(-\gamma\|x - z\|^2)$

Table 1: Some types of kernel used in non-linear kernel SVM

Implementation: We use the following hyperparameters to tune the SVR model:

- C: Normalization index. A lower C value leads to higher normalization power.
- ε: Deviation constant for SVR.
- $kernel$: Radial basis function is chosen to be the kernel instead of linear function.
- $gamma$: Radial basis function kernel-specific hyper-parameters.

3.3 Random Forest

3.3.1 How Decision tree works for regression?

Decision Trees follow a tree-like flowchart structure to show the predictions that result from a series of feature-based splits. It starts with a root node and ends with a decision made by leaves.

When a new data point is received, its attributes are pushed through the decision tree from the root to the nodes with corresponding attribute statistics. At each node, according to the value of the chosen attributes, the tree will decide which node will the data be passed on to. Usually, this path is considered based on the similarity of the data point to the attribute used to stem the tree - the branch with the highest similarities wins the data point. The data flow stops at a leaf of a decision tree, in which it receives its “decision” made by the decision tree instead of a pass.

The node split process identifies the optimal feature and threshold value to divide a node. It computes the CART cost function for various features k and thresholds t_k, then selects the feature and threshold that minimize the cost function for the split.

$$J(k, t_k) = \frac{m_{left}}{m} MSE_{left} + \frac{m_{right}}{m} MSE_{right},$$

$$where \begin{cases} MSE_{node} = \sum_{i \in node} (\hat{y}_{node} - y^i)^2 \\ \hat{y}_{node} = \frac{1}{m_{node}} \sum_{i \in node} y^i \end{cases}$$

3.3.2 Random Forest’s core

Ensemble Model:

Ensemble learning is a machine learning technique that enhances accuracy and resilience in forecasting by merging predictions from multiple models. It aims to mitigate errors or biases that may exist in individual models by leveraging the collective intelligence of the ensemble.

The underlying concept behind ensemble learning is to combine the outputs of diverse models to create a more precise prediction. By considering multiple perspectives and utilizing the strengths of different models, ensemble learning improves the overall performance of the learning system. This approach not only enhances accuracy but also provides resilience against uncertainties in the data. By effectively merging predictions from multiple models, ensemble learning has proven to be a powerful tool in various domains, offering more robust and reliable forecasts.

Bagging:

Bagging, also known as bootstrap aggregation, is an ensemble learning technique that combines the benefits of bootstrapping and aggregation to yield a stable model and improve the prediction performance of a machine-learning model.

In detail, each model is trained on a random subset of the data sampled with replacement, meaning that the individual data points can be chosen more than once. This random subset is known as a bootstrap sample. By training models on different bootstraps, bagging reduces the variance of the individual models. It also avoids overfitting by exposing the constituent models to different parts of the dataset.

The predictions from all the sampled models are then combined through simple averaging to make the overall prediction. This way, the aggregated model incorporates the strengths of the individual ones and cancels out their errors.

Random Forest's Core Ideas:

The random forest algorithm is an extension of the bagging method as it utilizes both bagging and feature randomness to create an uncorrelated forest of decision trees.

Feature randomness generates a random subset of features, which ensures low correlation among decision trees. This is a key difference between decision trees and random forests. While decision trees consider all the possible feature splits, random forests only select a subset of those features.

3.3.3 Implementation

We optimize the models' hyper-parameter based on the following parameters:

- $n\_estimators$: The number of trees in the forest
- $max\_features$: The number of features to consider when looking for the best split
- max_depth: The maximum depth of the tree
- $min\_samples\_split$: The minimum number of samples required to split an internal node
- $min\_samples\_leaf$: The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least $min\_samples\_leaf$ training samples in each of the left and right branches.

3.4 Gradient Boosting

The term "gradient boosting" refers to the process of enhancing the performance of a weak model by combining it with several other weak models to create a robust final model. It extends traditional boosting by framing the sequential addition of weak models as a gradient descent optimization over an objective function. In gradient boosting, the target for each instance is determined based on the gradient of the error with respect to the current prediction.

First, algorithm start with an initial prediction, often the mean of the target values for regression.

$$F_0(x) = \text{initial prediction}$$

Then, algorithm will iterative improve the model for M boosting rounds.

1. **Compute Residuals:** Calculate the residuals $r_i^{(m-1)}$ for each data points, which represent the errors of the current model $F_{m-1}(x)$

$$r_i^{(m-1)} = y_i - F_{m-1}(x_i)$$

2. **Fit a new model:** Train a new weak learner $h_m(x)$ to predict these residuals. This model aims to capture the errors of the previous model.

3. **Update the model:** Update the current model by adding the predictions of the new learner, scaled by a learning rate η:

$$F_m(x) = F_{m-1}(x) + \eta h_m(x)$$

The final prediction of the model is:

$$\hat{y}_i = F_M(x_i)$$

Gradient Boosted Decision Trees (GBDT) iteratively build an ensemble of shallow decision trees. While random forests use "bagging" to reduce variance and avoid overfitting, GBDT uses "boosting" to reduce bias and prevent underfitting.

3.4.1 XGBoost & LightGBM

XGBoost is a robust and scalable gradient boosting implementation aimed at optimizing both model accuracy and computational efficiency. Unlike traditional Gradient Boosted Decision Trees (GBDT), which construct trees sequentially, XGBoost builds trees in parallel and employs a level-wise approach. It assesses split quality by scanning gradient values and using partial sums to determine the most effective splits during training.

In contrast, LightGBM is another widely used gradient boosting framework with distinct features. LightGBM employs a leaf-wise tree growth strategy, which involves growing trees by expanding the leaf that offers the greatest reduction in error. This approach often results in deeper and less balanced trees. While LightGBM is highly efficient for large datasets, its method can lead to a higher risk of overfitting in small datasets.

3.4.2 Implementation:

We optimize the models' hyper-parameter based on the following parameters:

XGBoost:

- eta: learning rate in GBM, the step size shrinkage used in update to prevent overfitting
- $gamma$: specifies the minimum loss reduction required to make a split
- max_depth: the maximum depth of a tree
- min_child_weight: defines the minimum sum of weights of all observations required in a child
- $subsample$: denotes the fraction of observations to be randomly samples for each tree
- reg_alpha: L1 regularization term on weights
- $n\_estimators$: number of decision tree models used in the algorithm

LightGBM:

- $n\_estimato$: mentioned above
- max_depth: mentioned above
- num_leaves: main parameter to control the complexity of the tree model, should be set about 55% to 65% of 2^{max_depth}
- $feature\_fraction$: decide percentage of features to construct each tree, similar with $subsample$ on XGBoost
- $bagging_fraction$: decide percentage of observations used in the training data for each iteration.

3.5 Stacking Regressor

Stacking is an ensemble learning method that combines multiple regression models using a meta-regressor. I implement this approach with the **StackingCVRegressor** from the **mlxtend** library.

The **StackingCVRegressor** leverages out-of-fold predictions: the data is divided into k-folds, and in each of the k iterations, k-1 folds are used to train the base-level regressors. These models are then applied to the remaining 1 fold (which was not used for training) to generate predictions. These predictions are stacked and passed as input to the second-level regressor. After training is complete, the base-level regressors are retrained on the entire dataset to make the final predictions.

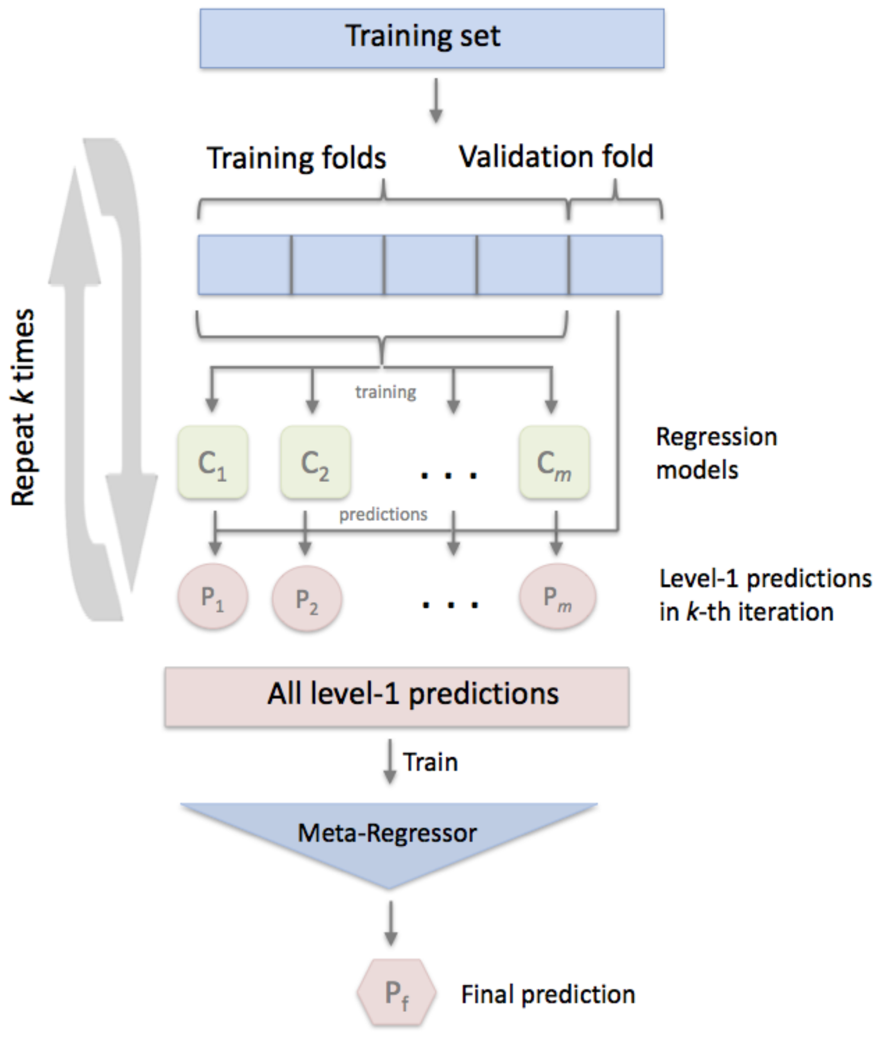

Figure 3: Stacked Regression

Implementation: I use the best-performing models from all the mentioned algorithms to build the final stacking model, with the meta-regressor being the top-performing one among them.

4 Experimental results

4.1 Model selection results

The results are split into two halves: the **left half** represents the **Bag of Words** preprocessing method, while the **right half** represents the **doc2vec** preprocessing method.

Notice that the results table includes the **F1-score** and **Accuracy**, which are not typical regression benchmarks. However, to provide additional insights, I rounded the values to integers, making them suitable for evaluation using these two metrics.

Abbreviations:

- LR: Linear Regressor
- SVR: Support Vector Regressor
- RF: Random Forest Regressor
- XGB: XGBoost Regressor
- LGBM: LightGBM Regressor
- Stacking: Stacking Cross-Validated Regressor

appceleratorstudio:

Bag of Words

	MSE	MAE	F1	Accuracy
LR	3.6851	**1.3265**	0.3431	**0.5087**
SVR	3.6329	1.4805	0.2803	0.2544
RF	3.3651	1.3496	0.3060	0.3310
XGB	3.5848	1.3500	**0.3508**	0.4390
LGBM	3.6135	1.3762	0.3166	0.4077
Stacking	**3.3413**	1.3759	0.2874	0.2822

doc2vec

	MSE	MAE	F1	Accuracy
LR	3.7961	1.4495	0.2908	0.3380
SVR	3.8448	1.5125	0.2691	0.2474
RF	3.8725	1.4742	0.2966	0.3031
XGB	3.7778	1.4321	0.3038	0.3589
LGBM	**3.7124**	**1.4063**	**0.3047**	**0.3798**
Stacking	3.7984	1.4316	0.3007	0.3554

aptanastudio:

Bag of Words

	MSE	MAE	F1	Accuracy
LR	27.4767	**3.3846**	0.0005	0.0130
SVR	29.8907	3.4002	0.0003	0.0130
RF	29.1247	3.4677	**0.1632**	**0.1169**
XGB	32.3785	3.6353	0.1039	0.0779
LGBM	29.9270	3.4598	0	0
Stacking	**27.4675**	**3.3846**	0.0005	0.0130

doc2vec

	MSE	MAE	F1	Accuracy
LR	28.4591	3.4476	0.0005	0.0130
SVR	29.8907	**3.4002**	0.0003	0.0130
RF	29.6765	3.4763	0.1415	0.1299
XGB	29.4241	3.5117	**0.1780**	**0.1429**
LGBM	29.3694	3.4859	0.0408	0.0390
Stacking	**28.3139**	3.4520	0.0006	0.0130

bamboo:

Bag of Words

	MSE	MAE	F1	Accuracy
LR	0.8298	0.7151	**0.1827**	**0.3514**
SVR	0.8554	**0.7049**	**0.1827**	**0.3514**
RF	0.8173	0.7124	**0.1827**	**0.3514**
XGB	0.8391	0.7214	**0.1827**	**0.3514**
LGBM	0.8428	0.7159	**0.1827**	**0.3514**
Stacking	**0.8160**	0.7113	**0.1827**	**0.3514**

doc2vec

	MSE	MAE	F1	Accuracy
LR	0.8428	0.7159	**0.1827**	**0.3514**
SVR	0.8631	**0.7026**	**0.1827**	**0.3514**
RF	0.8548	0.7248	**0.1827**	**0.3514**
XGB	**0.8391**	0.7214	**0.1827**	**0.3514**
LGBM	0.8731	0.7346	**0.1827**	**0.3514**
Stacking	**0.8391**	0.7214	**0.1827**	**0.3514**

clover:

Bag of Words

	MSE	MAE	F1	Accuracy
LR	4.0862	1.6842	0.0065	0.0556
SVR	4.3580	1.7176	0.0058	0.0556
RF	3.7604	1.4116	0.0755	0.1667
XGB	**3.1809**	**1.3687**	**0.1123**	**0.2222**
LGBM	4.5844	1.8178	0.0058	0.0556
Stacking	3.7174	1.3713	0.1961	0.2500

doc2vec

	MSE	MAE	F1	Accuracy
LR	4.5797	1.8160	0.0058	0.0556
SVR	4.3555	1.7164	0.0058	0.0556
RF	4.4065	**1.7083**	**0.1003**	**0.1667**
XGB	4.4716	1.7708	0.0058	0.0556
LGBM	4.5844	1.8178	0.0058	0.0556
Stacking	**4.3554**	1.7164	0.0058	0.0556

datamanagement:

Bag of Words

	MSE	MAE	F1	Accuracy
LR	64.2974	3.8291	0.1002	0.1191
SVR	62.3287	3.7603	0.1418	0.1414
RF	71.4102	**3.5437**	0.0900	0.1017
XGB	65.3111	3.9848	0.1199	0.1241
LGBM	93.0929	4.0668	0.0860	**0.1563**
Stacking	**62.2274**	3.8913	**0.1727**	0.1464

doc2vec

	MSE	MAE	F1	Accuracy
LR	82.3480	4.0923	0.1062	**0.1042**
SVR	**72.2202**	4.5094	0.0564	0.0596
RF	78.1436	4.2478	0.0968	0.0943
XGB	79.9471	4.3859	**0.1113**	0.1017
LGBM	89.1632	**4.0544**	0.0720	0.0868
Stacking	89.9221	4.6969	0.0772	0.0720

duracloud:

Bag of Words

	MSE	MAE	F1	Accuracy
LR	1.1878	0.7833	0.3781	0.4098
SVR	1.1615	0.8249	0.1763	0.3443
RF	1.1051	0.7657	**0.4095**	**0.4426**
XGB	1.2929	0.8416	0.3150	0.3607
LGBM	1.1808	0.8292	0.2101	0.3607
Stacking	**1.0930**	**0.7652**	0.4068	**0.4426**

doc2vec

	MSE	MAE	F1	Accuracy
LR	0.9451	0.6968	0.4044	**0.4754**
SVR	1.1745	0.8254	0.3200	0.3770
RF	1.0965	0.7945	0.3454	0.3770
XGB	1.1588	0.7895	0.3397	0.4098
LGBM	1.1127	0.7876	0.3386	0.4262
Stacking	**0.9969**	**0.7059**	**0.4112**	**0.4754**

jirasoftware:

Bag of Words

	MSE	MAE	F1	Accuracy
LR	7.4531	1.9263	0.0733	0.1429
SVR	7.5342	2.1449	0.0025	0.0357
RF	6.8756	1.8019	0.1265	**0.2500**
XGB	7.6232	2.0185	0.0350	0.0714
LGBM	7.7378	2.0697	0.0031	0.0357
Stacking	**6.8068**	**1.7552**	**0.1566**	0.2143

doc2vec

	MSE	MAE	F1	Accuracy
LR	7.1289	1.9280	0.0909	0.1429
SVR	7.2956	1.7884	0.1931	**0.2500**
RF	7.4734	2.1073	0.0025	0.0357
XGB	7.5055	2.1278	0.0025	0.0357
LGBM	7.4623	2.0998	0.0025	0.0357
Stacking	**6.7010**	**1.7366**	**0.2243**	**0.2500**

mesos:

Bag of Words

	MSE	MAE	F1	Accuracy
LR	2.2735	1.1233	0.1893	0.2564
SVR	2.3433	1.1488	**0.2711**	0.3333
RF	2.2977	**1.0939**	0.2646	**0.3590**
XGB	2.3755	1.1623	0.2372	0.2949
LGBM	2.4304	1.1521	0.2556	0.3462
Stacking	**2.2579**	1.1157	0.2076	0.2821

doc2vec

	MSE	MAE	F1	Accuracy
LR	2.5090	1.1874	0.2213	0.2949
SVR	2.6225	1.2353	0.2018	0.2564
RF	**2.3917**	1.1535	0.2472	0.3205
XGB	2.5232	1.1873	0.2438	0.3205
LGBM	2.3801	**1.1462**	0.2402	0.3269
Stacking	2.3975	1.1476	**0.2448**	**0.3333**

moodle:

Bag of Words

	MSE	MAE	F1	Accuracy
LR	99.4640	7.5848	**0.0169**	0.0086
SVR	62.5891	7.0800	0.0094	0.0086
RF	**42.7748**	**5.9883**	0.0052	0.0086
XGB	54.0043	6.7977	0	0
LGBM	55.6850	7.2413	0.0023	**0.0172**
Stacking	49.2742	6.1263	0.0054	0.0086

doc2vec

	MSE	MAE	F1	Accuracy
LR	**71.8456**	**7.9185**	0	0
SVR	92.5963	8.4975	0	0
RF	93.3487	8.8428	**0.0031**	**0.0086**
XGB	114.1811	9.6697	0	0
LGBM	72.5536	8.2308	0	0
Stacking	85.0151	8.3484	0	0

mule:

Bag of Words

	MSE	MAE	F1	Accuracy
LR	12.4946	2.7683	0.0634	0.0795
SVR	14.5248	2.8036	0	0
RF	13.6435	**2.6991**	**0.1084**	**0.1364**
XGB	14.1419	2.7794	0.0727	0.0795
LGBM	13.8248	2.7371	0.0338	0.0227
Stacking	**12.3859**	2.6999	0.0994	0.1136

doc2vec

	MSE	MAE	F1	Accuracy
LR	13.4977	2.7243	0.0824	0.0795
SVR	**11.5359**	**2.4911**	**0.1416**	**0.1250**
RF	13.2501	2.6985	0.0662	0.0909
XGB	13.5397	2.7408	0.0546	0.0568
LGBM	13.3719	2.7233	0.0428	0.0341
Stacking	12.3110	2.5793	0.1255	**0.1250**

mulestudio:

Bag of Words

	MSE	MAE	F1	Accuracy
LR	42.1508	4.4831	0.0935	0.0959
SVR	40.6806	**4.4133**	0.0875	**0.1507**
RF	40.7193	4.4375	**0.0990**	0.1096
XGB	41.4056	4.5553	0.0830	0.0822
LGBM	41.7452	4.4581	0.0897	0.1370
Stacking	**40.6775**	**4.4133**	0.0875	**0.1507**

doc2vec

	MSE	MAE	F1	Accuracy
LR	39.9823	4.2911	0.0736	0.1370
SVR	38.6703	4.2854	0.0715	0.1370
RF	40.2186	4.3303	0.0709	0.0685
XGB	40.0561	4.2996	**0.1226**	**0.1918**
LGBM	39.7961	4.2713	0.0795	0.1370
Stacking	**38.5156**	**4.2203**	0.0771	0.1370

springxd:

Bag of Words

	MSE	MAE	F1	Accuracy
LR	5.2025	1.5434	0.2044	0.2426
SVR	5.9621	1.5711	0.2419	**0.2787**
RF	5.0425	1.5355	0.2135	0.2393
XGB	**5.0329**	1.5562	0.1638	0.1836
LGBM	5.5155	1.6382	0.0685	0.1639
Stacking	5.1319	**1.5127**	**0.2453**	0.2656

doc2vec

	MSE	MAE	F1	Accuracy
LR	5.2591	**1.5778**	0.1886	0.2328
SVR	5.2351	1.6129	0.1227	0.1738
RF	5.3150	1.6285	0.1663	0.2033
XGB	5.4859	1.6403	0.1465	0.1967
LGBM	5.3629	1.6200	0.1358	0.2033
Stacking	**5.0293**	1.5938	**0.2186**	**0.2393**

talenddataquality:

Bag of Words

	MSE	MAE	F1	Accuracy
LR	10.7254	2.8065	**0.0823**	**0.1327**
SVR	13.4430	3.2158	0.0492	0.0531
RF	**9.2919**	2.6670	0.0440	0.0531
XGB	10.9050	2.8928	0.0353	0.0531
LGBM	11.4348	3.0448	0.0242	0.0708
Stacking	9.3193	**2.6417**	0.0554	0.0619

doc2vec

	MSE	MAE	F1	Accuracy
LR	10.3898	2.8470	**0.0590**	**0.0885**
SVR	16.0696	3.4160	0.0398	0.0531
RF	10.2852	**2.7799**	0.0563	0.0708
XGB	**10.2010**	2.8473	0.0504	**0.0885**
LGBM	10.4601	2.9276	0.0275	0.0708
Stacking	10.3419	2.9007	0.0360	0.0619

talendesb:

Bag of Words

	MSE	MAE	F1	Accuracy
LR	**1.2562**	0.7648	0.3330	0.3636
SVR	1.5648	0.7829	**0.4726**	**0.4805**
RF	1.4890	0.7543	0.3537	0.4156
XGB	1.4060	0.7772	0.4235	0.4416
LGBM	1.6462	0.8266	0.1481	0.3117
Stacking	1.2821	**0.7467**	0.4421	0.4545

doc2vec

	MSE	MAE	F1	Accuracy
LR	1.4468	0.7943	0.2226	0.3377
SVR	**1.3374**	**0.7581**	**0.4408**	**0.4675**
RF	1.5648	0.8278	0.3961	0.4026
XGB	1.6048	0.8734	0.3493	0.3506
LGBM	1.6410	0.8343	0.2121	0.3247
Stacking	1.6073	0.8222	0.4500	0.4545

titanium:

Bag of Words

	MSE	MAE	F1	Accuracy
LR	12.3972	2.1553	0.2150	0.3156
SVR	12.7608	2.1700	0.2381	**0.4089**
RF	12.1467	2.0023	0.2344	0.2489
XGB	12.1941	2.2021	0.1585	0.1200
LGBM	12.4178	2.1461	0.1965	0.2489
Stacking	**11.9501**	**1.9977**	**0.2529**	0.2444

doc2vec

	MSE	MAE	F1	Accuracy
LR	13.2224	**2.1908**	**0.2095**	**0.3538**
SVR	13.2586	2.3047	0.1831	0.2311
RF	13.3797	2.3059	0.1547	0.1792
XGB	**13.1052**	2.2646	0.1756	0.2217
LGBM	13.3013	2.2466	0.1897	0.2406
Stacking	13.3728	2.2764	0.1924	0.2217

usergrid:

Bag of Words

	MSE	MAE	F1	Accuracy
LR	3.5212	1.3929	0.1855	0.2727
SVR	3.5377	1.3614	**0.1939**	**0.3636**
RF	3.4945	1.3870	0.1636	0.2727
XGB	**3.4284**	**1.3580**	0.1636	0.2727
LGBM	3.5161	1.3708	**0.1939**	**0.3636**
Stacking	3.5149	1.3796	0.1818	0.3030

doc2vec

	MSE	MAE	F1	Accuracy
LR	3.5101	1.3636	0.1939	0.3636
SVR	3.4999	1.3550	0.1939	0.3636
RF	3.5490	1.3619	0.1939	0.3636
XGB	**3.1949**	**1.2728**	0.2217	0.3333
LGBM	3.3491	1.2865	0.2551	0.3636
Stacking	3.3810	1.3252	**0.2753**	**0.3939**

4.2 General observations

There is no single "best" algorithm that universally excels across all datasets. The effectiveness of an algorithm can vary significantly depending on the specific characteristics and intricacies of each dataset.

However, when focusing solely on Mean Squared Error (MSE) and Mean Absolute Error

(MAE) metrics, it becomes evident that Stacking models generally perform well across a majority of datasets. Notably, Stacking models have demonstrated superior performance on datasets such as the jirasoftware dataset, the springxd-doc2vec dataset, and the titanium-BoW dataset. Their ability to combine predictions from multiple models often leads to a better accuracy and robustness in these cases.

The performance of other algorithms can be less predictable and varies significantly depending on the dataset. For instance, complex models such as Random Forest Regressor, XGBoost Regressor, and LightGBM Regressor may yield impressive results on some datasets but fall short on others. Similarly, simpler models like Linear Regressor and Support Vector Regressor show variable performance across different datasets. This inconsistency can often be attributed to the specific nature of datasets and the preprocessing techniques applied.

In terms of classification metrics such as F1-score and Accuracy, the results across evaluations have been notably low, failing to exceed a score of 0.5. This outcome is somewhat expected, given that the approach under consideration is fundamentally a regression problem rather than a classification problem. The mismatch between the problem's nature and the chosen metrics likely contributes to these lower performance scores.

4.3 Compare between Bag of Words and doc2vec

In this project, two tokenization methods were used: **Bag of Words** and **doc2vec**. The **doc2vec** preprocessing method produces a fixed-length feature vector of 128 dimensions, with the transformed information encapsulated in a dense representation. In contrast, the **Bag of Words** method typically results in a much larger set of features. While this can provide more granular information, it also comes with increased computational costs.

Based on the evaluation results, the differences between these two methods are generally minimal. In fact, **doc2vec** often shows higher error rates compared to **BoW** in several intances. However, **doc2vec** offer the advantage of consistent fixed-length feature vector, which is usually smaller than the feature sets produced by **BoW**. This fixed length significantly reduces computational overhead.

Overall, despite the occasional higher error rates with **doc2vec**, its efficiency in terms of computational cost and consistent feature size supports its overall superiority compared to Bag of Words method.

5 Conclusion and Further Enhancements

In this project, I investigated the story point datasets - which are my first Software Engineer datasets using data exploratory and Machine Learning tools. I showed some major traits of the dataset, analysed its properties, before performing the necessary preprocessing steps to make the data easier to model. Later, I deployed various Machine Learning models to try to establish a learnable program that can return the story point of certain tasks.

Based on our investigations, we have some conclusions about the dataset and the functionality of Machine Learning models when applied to it, as well as some ways we can enhance our models in the future to address the problems:

- Although the range of story points in these datasets spans from 1 to 100, the number of unique values is limited. If we treat these values as discrete classes, the problem can be approached as a classification task rather than a regression one. While this approach may have drawbacks due to the ordinal nature of the labels, it still holds promise.
- Despite using complex model like ensemble model, the results seem not to be positive. Therefore, in order to increase the accuracy of our work, applying Deep Learning models specified for NLP-related tasks, like Long-Short Term Memory (LSTM), Transformer models as BERTs, ... is an efficient movement.

6 Acknowledgement

Lastly, I would like to deliver our utmost appreciation to Mentor. Dang Yen Trang from RISE lab, BKAI, Hanoi University of Science and Technology, who was also the mentor of this project, for equipping us with the necessary knowledge and skills which are necessary to the process of establishing this project.

7 References

- Khoat Than. (2024, Semester 2). IT3190E - Machine Learning. Lecture presented at Hanoi University of Science and Technology, March 2024.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Blondel, M. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12(Oct), 2825-2830.
- Tiep, Vu Huu, "Machine Learning cơ bản", published March 27^{th}, 2018.
- Tianqi Chen, "XGBoost: A Scalable Tree Boosting System", published June 10^{th} , 2016
- Nihal Desai and Vatsal Patel, "Linear Decision Tree Regressor: Decision Tree Regressor Combined with Linear Regressor", published July, 2016
- Debasish Basak and Srimanta Pal, "Support Vector Regression", published November, 2007